

CE266: SOFTWARE ENGINEERING

December 2023 – April 2024

UNIT 4

Requirement Analysis and Specification

Content



Understanding the Requirement

Requirement Modeling

Requirement Specification (SRS)

Requirement Analysis and Requirement Elicitation

Requirement Engineering

Understanding the Requirement

■ Requirement Engineering:

1. **Inception**—ask a set of questions that establish ...
 - basic understanding of the problem
 - the people who want a solution
 - the nature of the solution that is desired, and
 - the effectiveness of preliminary communication and collaboration between the customer and the developer
2. **Elicitation**—elicit requirements from all stakeholders
3. **Elaboration**—create an analysis model that identifies data, function and behavioral requirements
4. **Negotiation**—agree on a deliverable system that is realistic for developers and customers

4. Specification—can be any one (or more) of the following:

- A written document
- A set of models
- A formal mathematical
- A collection of user scenarios (use-cases)
- A prototype

5. Validation—a review mechanism that looks for

- errors in content or interpretation
- areas where clarification may be required
- missing information
- inconsistencies (a major problem when large products or systems are engineered)
- conflicting or unrealistic (unachievable) requirements.

6. Requirements management

Inception

- Identify stakeholders
 - “who else do you think I should talk to?”
- Recognize multiple points of view
- Work toward collaboration
- The first questions
 - Who is behind the request for this work?
 - Who will use the solution?
 - What will be the economic benefit of a successful solution
 - Is there another source for the solution that you need?

Eliciting Requirements

- meetings are conducted and attended by both software engineers and customers
- rules for preparation and participation are established
- an agenda is suggested
- a "facilitator" (can be a customer, a developer, or an outsider) controls the meeting
- a "definition mechanism" (can be work sheets, flip charts, or wall stickers or an electronic bulletin board, chat room or virtual forum) is used
- the goal is
 - to identify the problem
 - propose elements of the solution
 - negotiate different approaches, and
 - specify a preliminary set of solution requirements

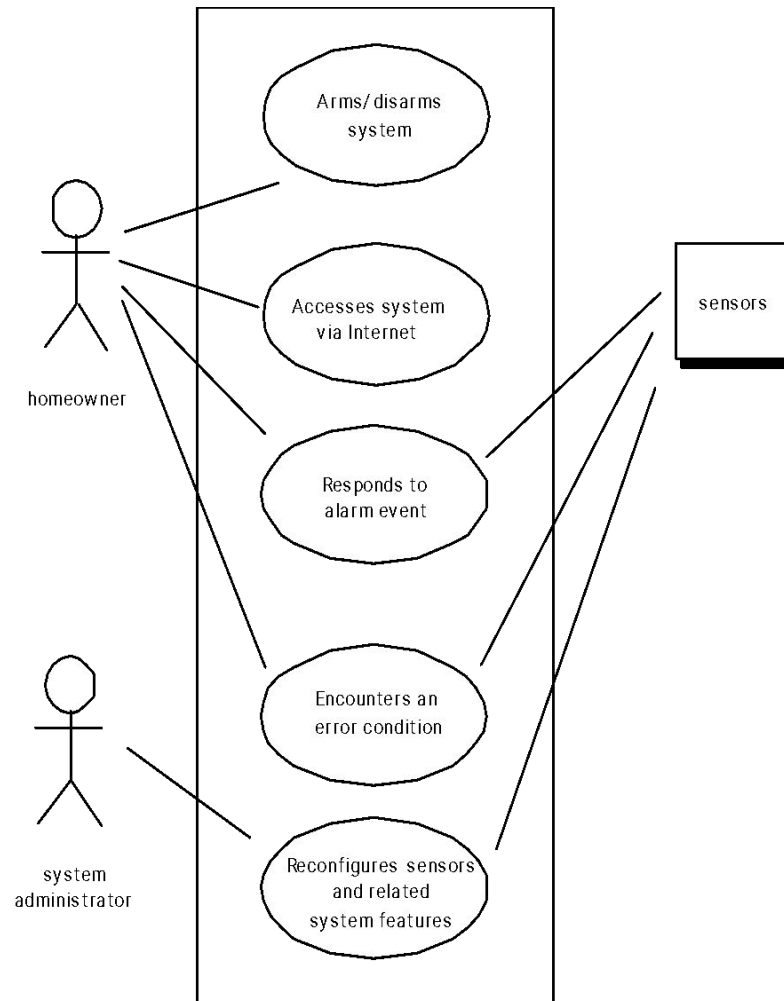
Elaboration - Building the Analysis Model

- Elements of the analysis model
 - Scenario-based elements
 - Functional—processing narratives for software functions
 - Use-case—descriptions of the interaction between an “actor” and the system
 - Class-based elements
 - Implied by scenarios
 - Behavioral elements
 - State diagram
 - Flow-oriented elements
 - Data flow diagram

Use-Cases

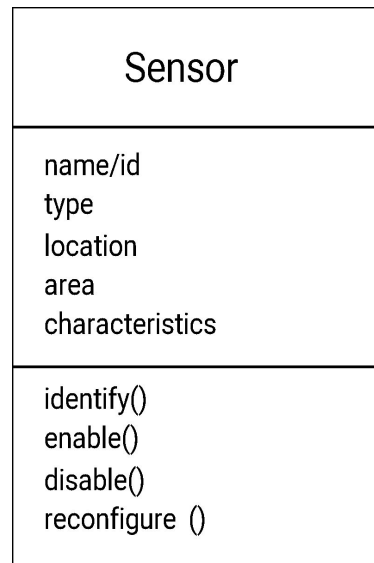
- A collection of user scenarios that describe the thread of usage of a system
- Each scenario is described from the point-of-view of an “actor”—a person or device that interacts with the software in some way
- Each scenario answers the following questions:
 - Who is the primary actor, the secondary actor (s)?
 - What are the actor’s goals?
 - What preconditions should exist before the story begins?
 - What main tasks or functions are performed by the actor?
 - What extensions might be considered as the story is described?
 - What variations in the actor’s interaction are possible?
 - What system information will the actor acquire, produce, or change?
 - Will the actor have to inform the system about changes in the external environment?
 - What information does the actor desire from the system?
 - Does the actor wish to be informed about unexpected changes?

Use-Case Diagram

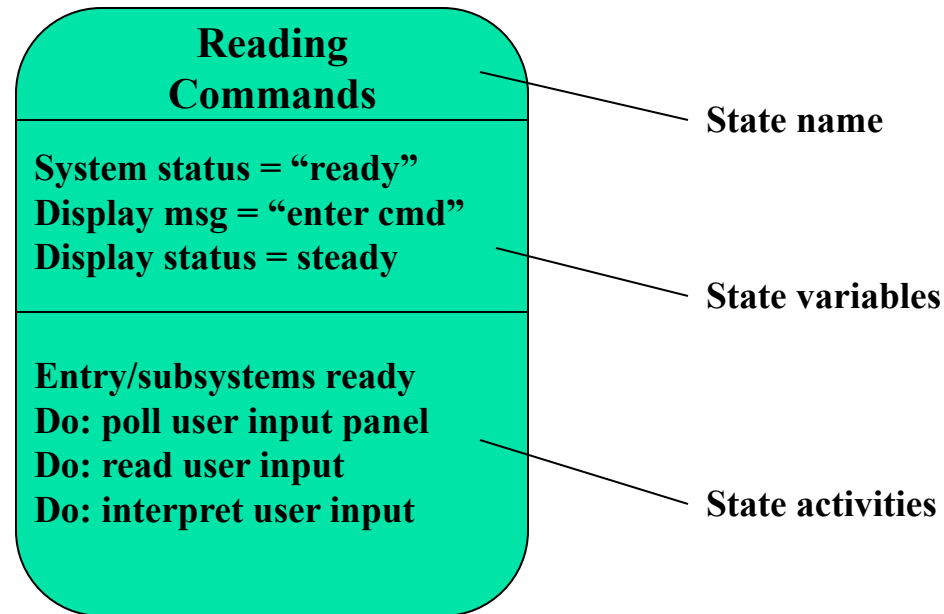


Class Diagram

From the *SafeHome* system ...



State Diagram



Negotiating Requirements

- Identify the key stakeholders
 - These are the people who will be involved in the negotiation
- Determine each of the stakeholders “win conditions”
 - Win conditions are not always obvious
- Negotiate
 - Work toward a set of requirements that lead to “win-win”

Validating Requirements

- Is each **requirement consistent with the overall objective** for the system/product?
- Have all **requirements been specified at the proper level of abstraction**? That is, do some requirements provide a level of technical detail that is inappropriate at this stage?
- Is the **requirement really necessary** or does it represent an add-on feature that may not be essential to the objective of the system?
- Is each **requirement bounded and unambiguous**?
- Does each requirement have attribution? That is, **is a source (generally, a specific individual) noted for each requirement**?
- Do any **requirements conflict with other requirements**?

Contd...

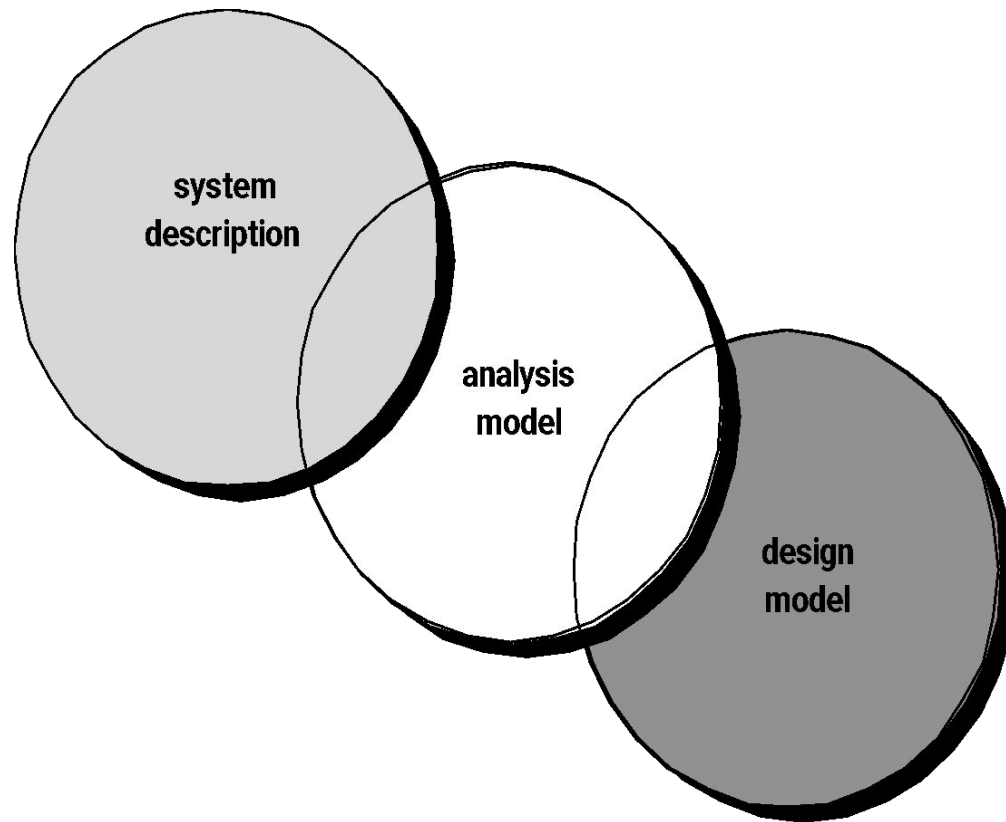
- Is each **requirement achievable** in the technical environment that will house the system or product?
- Is each **requirement testable, once implemented**?
- Does the **requirements model properly reflect the information, function and behavior of the system to be built.**
- Has the requirements model been “partitioned” in a way that exposes progressively more **detailed information about the system.**
- Have **requirements patterns been used to simplify the requirements model.** Have all patterns been properly validated? Are all patterns consistent with customer requirements?

Requirement Modeling

■ Requirement Analysis:

- specifies software's **operational characteristics**
 - indicates **software's interface with other system elements**
 - establishes **constraints** that software must meet
- Requirements analysis allows the software engineer (called an *analyst* or *modeler* in this role) to:
- elaborate on basic requirements established during earlier requirement engineering tasks
 - build models that depict user scenarios, functional activities, problem classes and their relationships, system and class behavior, and the flow of data as it is transformed.

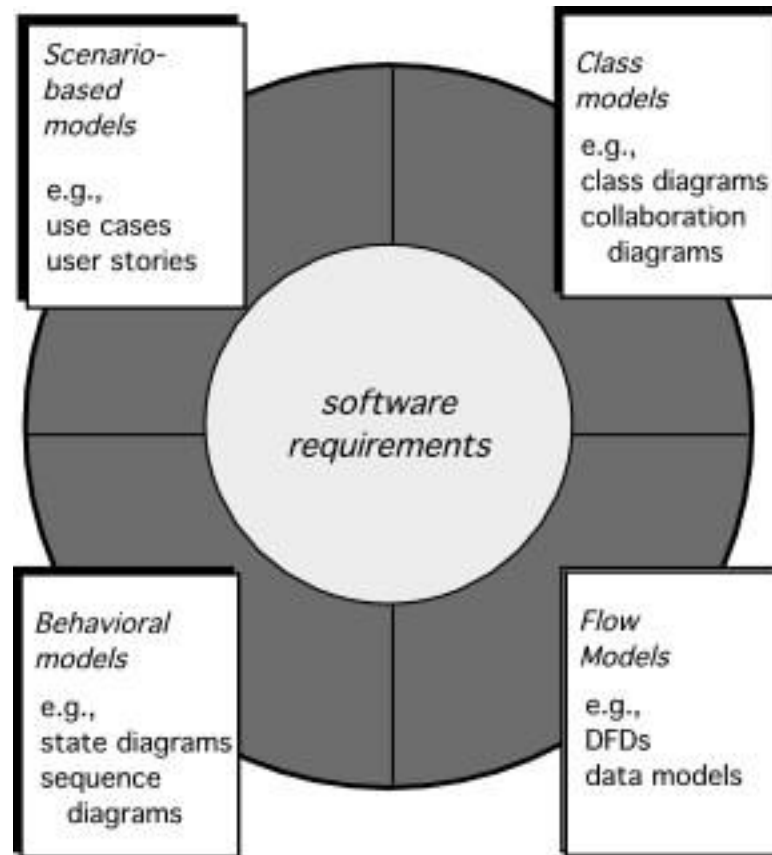
A Bridge



Rules of Thumb

- The model should focus on **requirements that are visible within the problem** or business domain. The level of abstraction should be relatively high.
- Each element of the analysis model should add to an **overall understanding of software requirements** and provide insight into the information domain, function and behavior of the system.
- Delay consideration of infrastructure and other non-functional models until design.
- **Minimize coupling** throughout the system.
- Be certain that the analysis model **provides value to all stakeholders**.
- Keep the **model as simple as it can be**.

Elements of Requirements Analysis



Scenario-Based Modeling

“[Use-cases] are simply an aid to defining what exists outside the system (actors) and what should be performed by the system (use-cases).” Ivar Jacobson

- (1) What should we write about?**
- (2) How much should we write about it?**
- (3) How detailed should we make our description?**
- (4) How should we organize the description?**

What to Write About?

- **Inception and elicitation**—provide you with the information you'll need to begin writing use cases.
- **Requirements gathering meetings, QFD (Quality Function Deployment), and other requirements engineering mechanisms** are used to
 - identify stakeholders
 - define the scope of the problem
 - specify overall operational goals
 - establish priorities
 - outline all known functional requirements, and
 - describe the things (objects) that will be manipulated by the system.
- To begin developing a set of use cases, **list the functions or activities performed by a specific actor**.

How Much to Write About?

- As further conversations with the stakeholders progress, the requirements gathering team develops use cases for each of the functions noted.
- In general, use cases are written first in an informal narrative fashion.
- If more formality is required, the same use case is rewritten using a structured format similar to the one proposed.

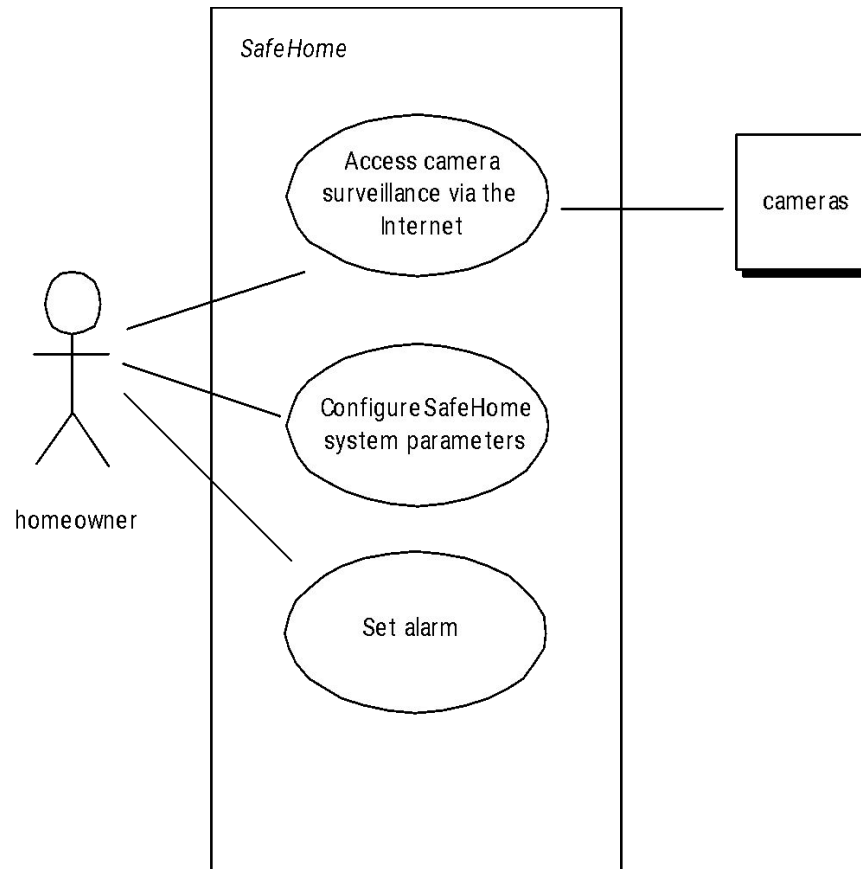
Use-Cases

- a scenario that describes a “thread of usage” for a system
- *actors* represent roles people or devices play as the system functions
- *users* can play a number of different roles for a given scenario

Developing a Use-Case

- What are the main tasks or functions that are performed by the actor?
- What system information will the the actor acquire, produce or change?
- Will the actor have to inform the system about changes in the external environment?
- What information does the actor desire from the system?
- Does the actor wish to be informed about unexpected changes?

Use-Case Diagram



Data Modeling

- examines data objects independently of processing
- focuses attention on the data domain
- creates a model at the customer's level of abstraction
- indicates how data objects relate to one another

Data Objects and Attributes

A data object contains a set of attributes that act as an aspect, quality, characteristic, or descriptor of the object

object: automobile

attributes:

make

model

body type

price

options code

CRC Models

- *Class-responsibility-collaborator (CRC) modeling* [Wir90] provides a simple means for identifying and organizing the classes that are relevant to system or product requirements. Ambler [Amb95] describes CRC modeling in the following way:
 - A CRC model is really a collection of standard index cards that represent classes. The cards are divided into three sections. Along the top of the card you write the name of the class. In the body of the card you list the class responsibilities on the left and the collaborators on the right.

CRC Modeling

Class: FloorPlan	
Description:	
Responsibility:	Collaborator:
defines floor plan name/type	
manages floor plan positioning	
scales floor plan for display	
scales floor plan for display	
incorporates walls, doors and windows	Wall
shows position of video cameras	Camera

Requirement Specification (SRS)

- It contains a **complete information** description, a detailed functional description, a representation of system behaviour, an indication of performance requirements and design constraints, appropriate validation criteria, and other information pertinent to requirements.
- Software requirement specification (SRS) is a **document that completely describes what the proposed software** should do without describing how software will do it.
- The basic goal of the requirement phase is to produce the SRS, Which describes the **complete behaviour** of the proposed software.
- SRS is also **helping the clients to understand their own needs**.

Characteristics of an SRS

- Software requirements specification should be **accurate, complete, efficient, and of high quality, so that it does not affect the entire project plan**. An SRS is said to be of high quality when the developer and user easily understand the prepared document. Other characteristics of SRS are discussed below.
- **Correct**
 - SRS is correct when all user requirements are stated in the requirements document.
 - The **stated requirements should be according to the desired system**.
 - This implies that **each requirement is examined to ensure that it (SRS) represents user requirements**.
 - Note that there is **no specified tool** or procedure to assure the correctness of SRS. Correctness ensures that all specified requirements are performed correctly.

Contd...

- **Unambiguous**

- SRS is unambiguous when every **stated requirement has only one interpretation**.
- This implies that each requirement is uniquely interpreted.
- In case there is a **term used with multiple meanings, the requirements document should specify the meanings in the SRS so that it is clear and easy to understand**

- **Complete**

- SRS is complete when the **requirements clearly define what the software is required to do**.
- This includes all the **requirements related to performance, design and functionality**.

- **Ranked for importance/stability**

- All requirements are not equally important, hence each requirement is identified to make differences among other requirements.
- For this, it is essential to clearly identify each requirement. Stability implies the probability of changes in the requirement in future.

- **Modifiable**

- The requirements of the user can change, hence requirements document should be created in such a manner that those changes can be modified easily, consistently maintaining the structure and style of the SRS.

- **Traceable**

- SRS is traceable when the **source of each requirement is clear and facilitates the reference of each requirement in future.**
- For this, **forward tracing and backward tracing are used.**
- **Forward tracing** implies that each requirement should be traceable to **design and code** elements.
- **Backward tracing** implies defining **each requirement explicitly referencing its source.**

- **Verifiable**

- SRS is verifiable when the specified requirements can be verified with a **cost-effective** process to check whether the final software meets those requirements.
- The requirements are verified with the help of reviews. Note that unambiguity is

■ Consistent

- SRS is consistent when the subsets of **individual requirements defined do not conflict with each other.**
- For example, there can be a case when different requirements can use different terms to refer to the same object.
- There can be logical or temporal conflicts between the specified requirements and some requirements whose logical or temporal characteristics are not satisfied.
- For instance, a requirement states that an event 'a' is to occur before another event 'b'. But then another set of requirements states (directly or indirectly by transitivity) that event 'b' should occur before event 'a'.

Attributes of Bad SRS Documents

1. **Over-Specification** – occurs when the analyst tries to address the “how to” aspects in the SRS document
2. **Forward References** – do not refer to the aspects that are discussed much later in the SRS document. It reduces the readability of the specification.
3. **Wishful Thinking** - this concerns to the description of aspects that would be difficult to implement.
4. **Noise** – presence of material not directly relevant to the software development process. Example: in the register customer function, suppose the analyst writes that the customer registration department is managed by clerks who report for work between 8:00 AM and 5:00 PM, 7 days a week. This information can be called noise as it would hardly be of any use to the software developers

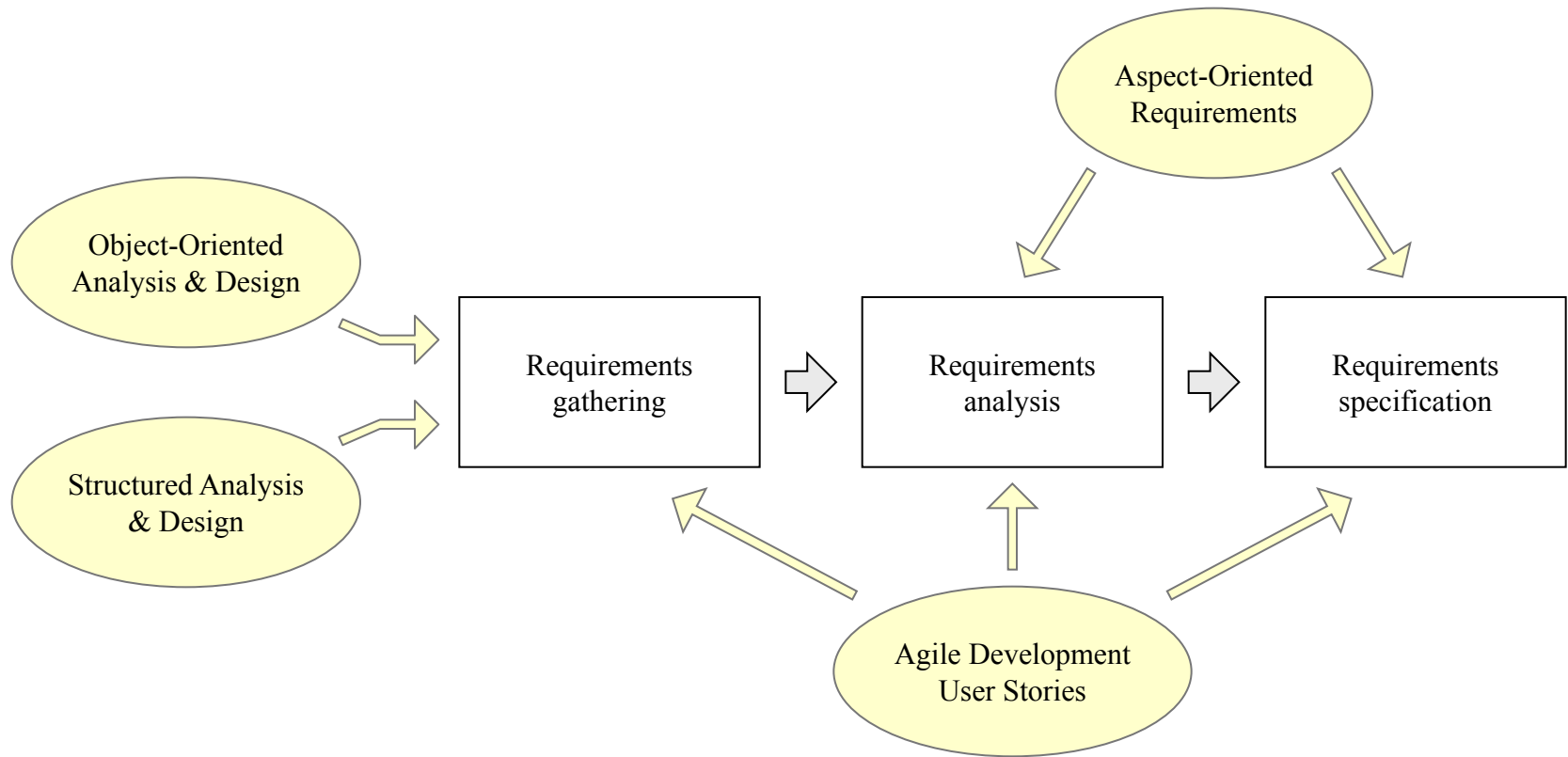
Functional and Non-Functional Requirements

- **Functional requirements**

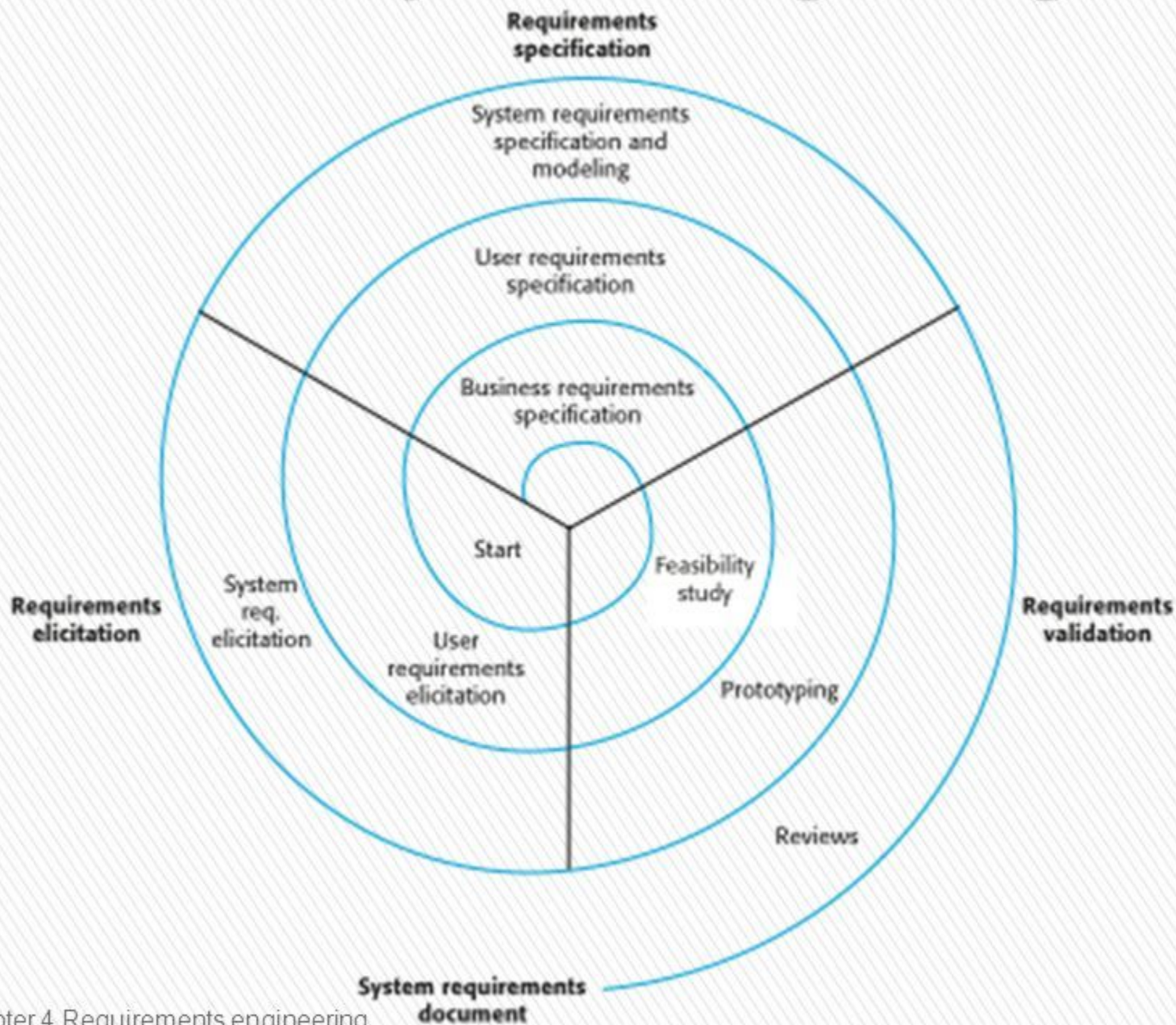
- These describe the functionality of a system -- how a system should react to a particular set of inputs and what should be the corresponding output.
- Given a problem statement, the functional requirements could be identified by focusing on the following points:
 - **Identify the high level functional requirements** simply from the conceptual understanding of the problem. For example, a Library Management System, apart from anything else, should be able to issue and return books.
 - **Identify the cases where an end user gets some meaningful work done by using the system.** For example, in a digital library a user might use the "Search Book" functionality to obtain information about the books of his interest.
 - If we consider the system as a black box, there would be some inputs to it, and some output in return. This black box defines the **functionalities of the system**. For example, to search for a book, user gives title of the book as input and get the book details and location as the output.
 - **Any high level requirement identified could have different sub-requirements.** For example, "Issue Book" module could behave differently for different class of users, or for a particular user who has issued the book thrice consecutively.

- **Non-Functional requirements (NFR)**
 - They are **not directly related what functionalities are expected from the system.**
 - However, NFRs could typically define how the system should behave under certain situations.
 - For example, a NFR could say that the system should work with 128MBRAM.

Requirements Process



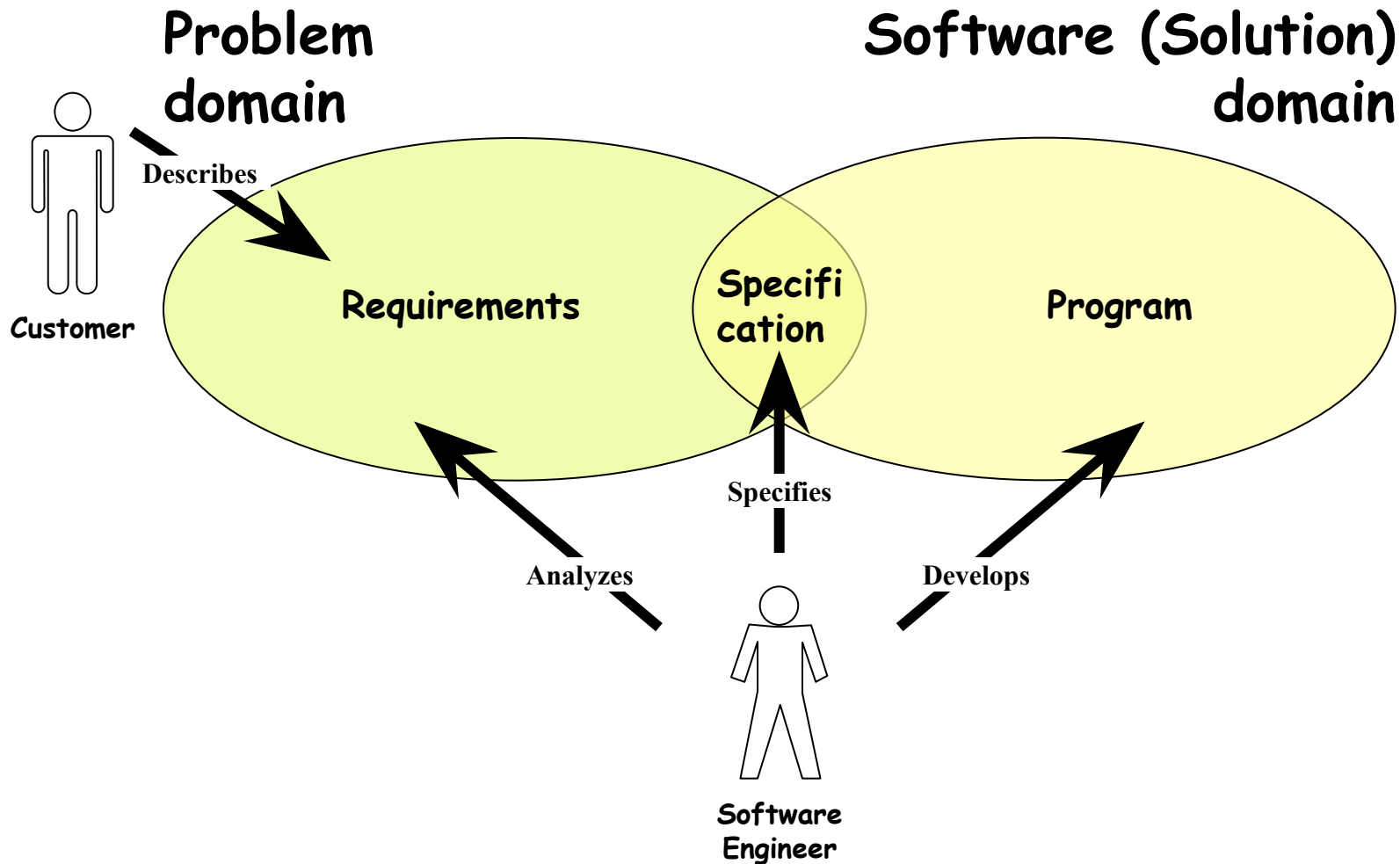
A spiral view of the requirements engineering process



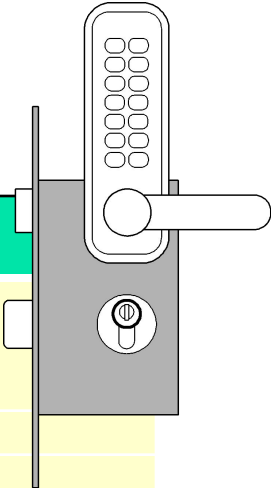
Requirements Engineering Components

- Requirements gathering
 - (a.k.a. “requirements elicitation”) helps the customer to define what is required: what is to be accomplished, how the system will fit into the needs of the business, and how the system will be used on a day-to-day basis
- Requirements analysis
 - refining and modifying the gathered requirements
- Requirements specification
 - documenting the system requirements in a semiformal or formal manner to ensure clarity, consistency, and completeness

Requirements and Specification



Example System Requirements



Identifier	Priority	Requirement
REQ1	5	The system shall keep the door locked at all times, unless commanded otherwise by authorized user. When the lock is disarmed, a countdown shall be initiated at the end of which the lock shall be automatically armed (if still disarmed).
REQ2	2	The system shall lock the door when commanded by pressing a dedicated button.
REQ3	5	The system shall, given a valid key code, unlock the door and activate other devices.
REQ4	4	The system should allow mistakes while entering the key code. However, to resist “dictionary attacks,” the number of allowed failed attempts shall be small, say three, after which the system will block and the alarm bell shall be sounded.
REQ5	2	The system shall maintain a history log of all attempted accesses for later review.
REQ6	2	The system should allow adding new authorized persons at runtime or removing existing ones.
REQ7	2	The system shall allow configuring the preferences for device activation when the user provides a valid key code, as well as when a burglary attempt is detected.
REQ8	1	The system should allow searching the history log by specifying one or more of these parameters: the time frame, the actor role, the door location, or the event type (unlock, lock, power failure, etc.). This function shall be available over the Web by pointing a browser to a specified URL.
REQ9	1	The system should allow filing inquiries about “suspicious” accesses. This function shall be available over the Web.

- Problem: Requirements prioritization.
- See how solved in **agile methods**.

Acceptance Tests

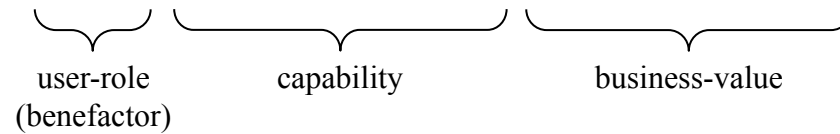
- An **acceptance test** specifies a set of scenarios for determining whether the finished system meets the customer requirements
- An **acceptance test case** specifies, for a given “situation” or “context” (defined by current system inputs), the output or behavior the system will produce in response

Problem: Requirements Prioritization

- When prioritizing requirements, “important” and “urgent” aspects can be confused
- Also, it is difficult to assign a numeric value of priority to each requirement
 - It requires more mental effort than just rank-ordering the requirements in a linear sequence

User Stories

As a tenant, I can unlock the doors to enter my apartment.



- Similar to system requirements, but focus on the user benefits, instead on system features.
- Preferred tool in **agile methods**.

Example User Stories

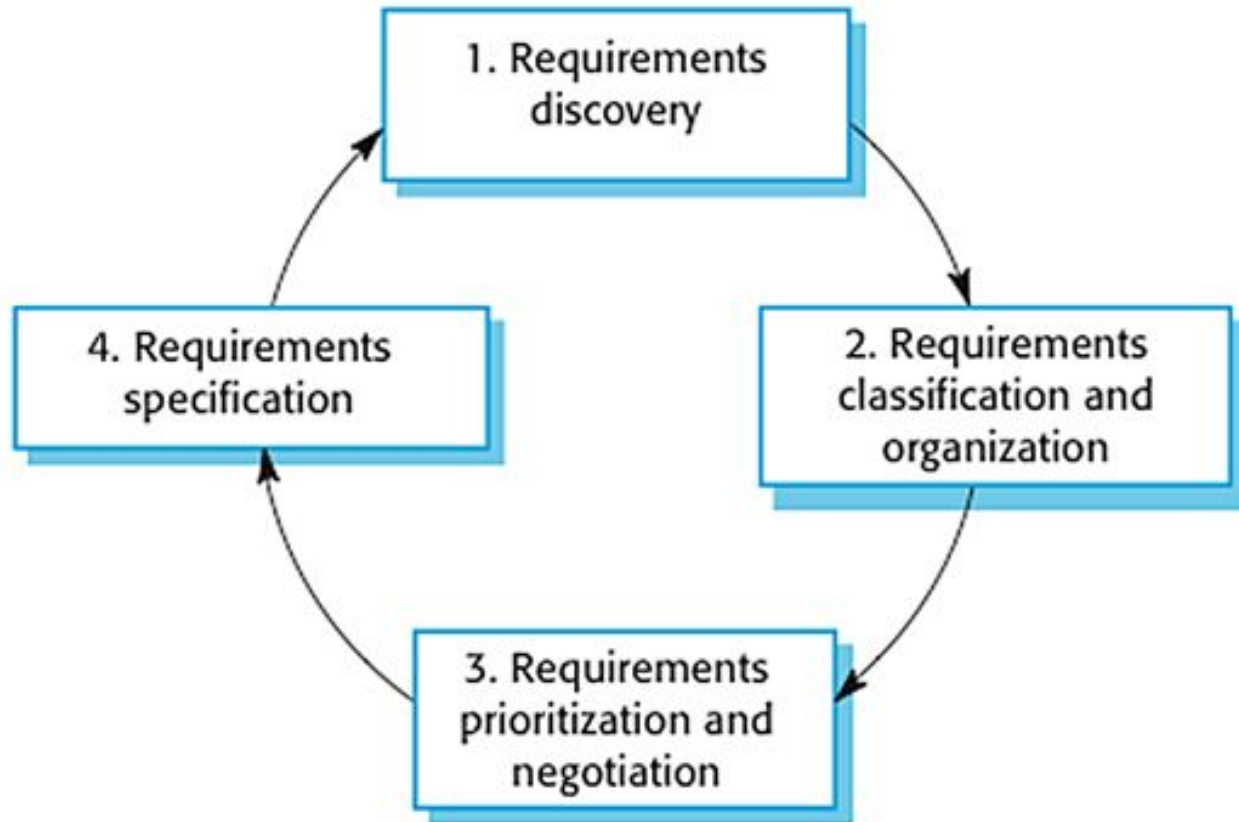
Identifier	User Story	Size
ST-1	As an authorized person (tenant or landlord), I can keep the doors locked at all times.	4 points
ST-2	As an authorized person (tenant or landlord), I can lock the doors on demand.	3 pts
ST-3	The lock should be automatically locked after a defined period of time.	6 pts
ST-4	As an authorized person (tenant or landlord), I can unlock the doors. (Test: Allow a small number of mistakes, say three.)	9 points
ST-5	As a landlord, I can at runtime manage authorized persons.	10 pts
ST-6	As an authorized person (tenant or landlord), I can view past accesses.	6 pts
ST-7	As a tenant, I can configure the preferences for activation of various devices.	6 pts
ST-8	As a tenant, I can file complaint about “suspicious” accesses.	6 pts

- Note no priorities for user stories.
- Story priority is given by its order of appearance on the work backlog (described next)
- Size points (last column) will be described later

Requirements Analysis Activities

- Not only refinement of customer requirements, but also feasibility and how realistic
- Needs to identify the points where **business policies** need to be applied.
- Explicit identification of business policies (BP) is important for two reasons:
 1. Making the need for BP explicit allows involving other stakeholders in decision about the solutions to adopt—Helps to involve others, particularly the customer, in decision making about each policy to adopt
 2. Helps to anticipate potential future changes in the policies, so mechanisms can be implemented in the code that localize the future changes and allow quick substitution of business policies

Requirements Elicitation



Requirements Elicitation Techniques

- Interviewing
 - Open interviews – there are no predefined agendas
 - Closed interviews – stakeholders answers a predefined set of questions
- Ethnography – watch people doing their job to see what artifacts they use, how they use them and so on.
- Stories and scenarios

Types of Requirements

- Functional Requirements
- Non-functional requirements (or quality requirements)
 - FURPS+
 - Functionality (security), Usability, Reliability, Performance , Supportability
- On-screen appearance requirements

On-screen Appearance Requirements

- Do not waste your time and your customer's time by creating elaborate screen shots with many embellishments, coloring, shading, etc., that serves only to distract attention from most important aspects of the interface
- Hand-drawing the proposed interface forces you to ***economize*** and focus on the most important features
- Only when there is a consensus that a good design is reached, invest effort to prototype the interface

Tools for Requirements Eng.

- Tools, such as user stories and use cases, used for:
 - Determining what exactly the user needs ("requirements analysis")
 - Writing a description of what system will do ("requirements specification")
- Difficult to use the same tool for different tasks (analysis vs. specification)

Acceptance Tests

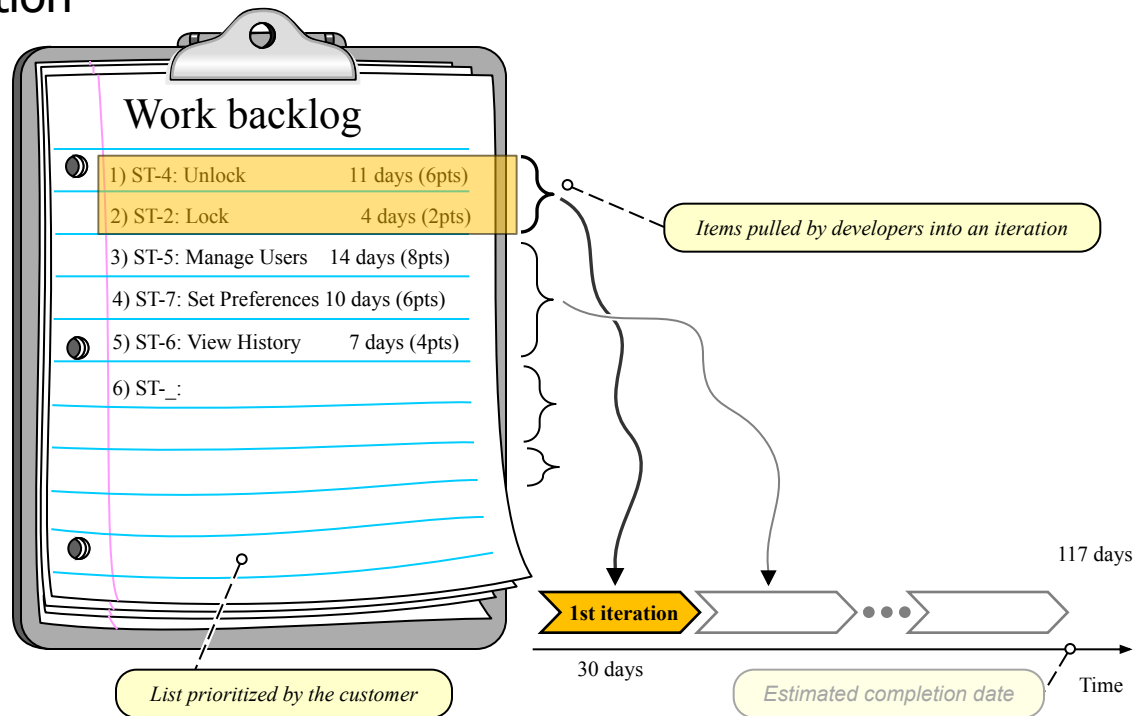
- Means of assessing that the requirements are met as expected
- Conducted by the customer
- An acceptance test describes whether the system will pass or fail the test, given specific input values
- Cannot ever guarantee 100% coverage of all usage scenarios, but *systematic approach* can increase the degree of coverage

Example User Stories

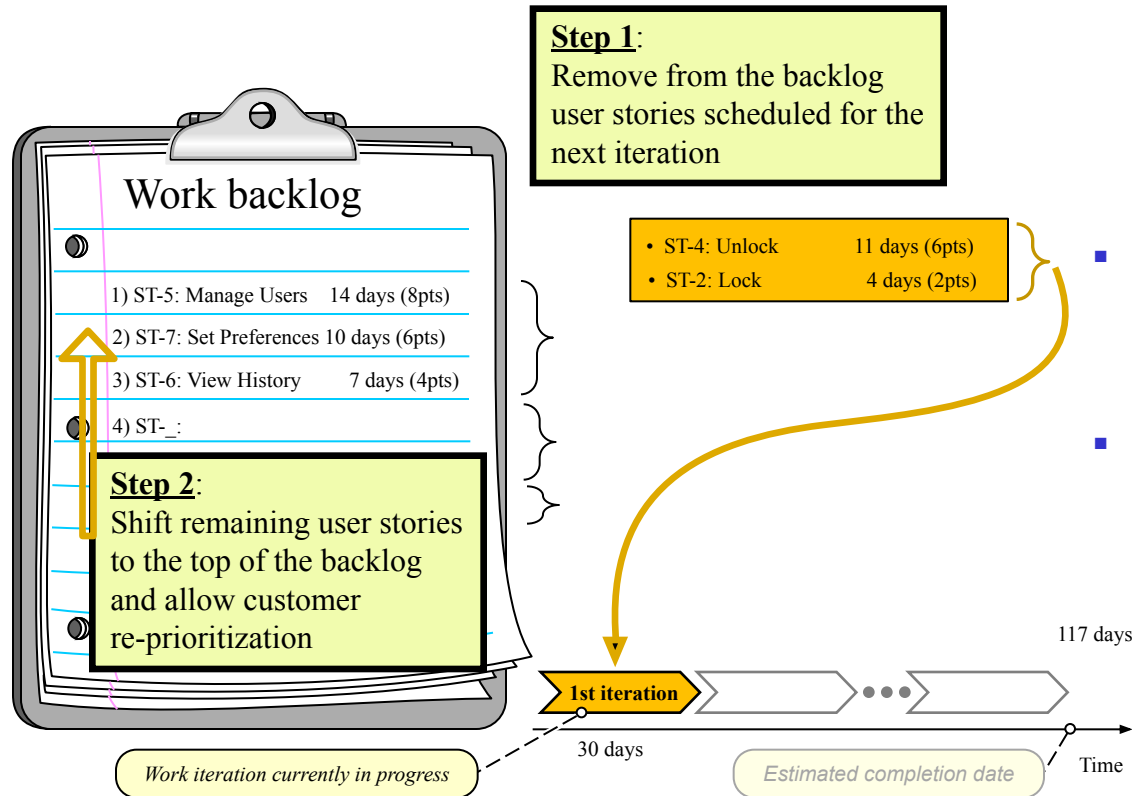
Identifier	User Story	Size
ST-1	As an authorized person (tenant or landlord), I can keep the doors locked at all times.	4 points
ST-2	As an authorized person (tenant or landlord), I can lock the doors on demand.	3 pts
ST-3	The lock should be automatically locked after a defined period of time.	6 pts
ST-4	As an authorized person (tenant or landlord), I can unlock the doors. (Test: Allow a small number of mistakes, say three.)	9 points
ST-5	As a landlord, I can at runtime manage authorized persons.	10 pts
ST-6	As an authorized person (tenant or landlord), I can view past accesses.	6 pts
ST-7	As a tenant, I can configure the preferences for activation of various devices.	6 pts
ST-8	As a tenant, I can file complaint about “suspicious” accesses.	6 pts

Agile Prioritization of Work

- Instead of assigning priorities, the customer creates an ordered list of user stories
- Developers simply remove the top list items and work on them in the next iteration

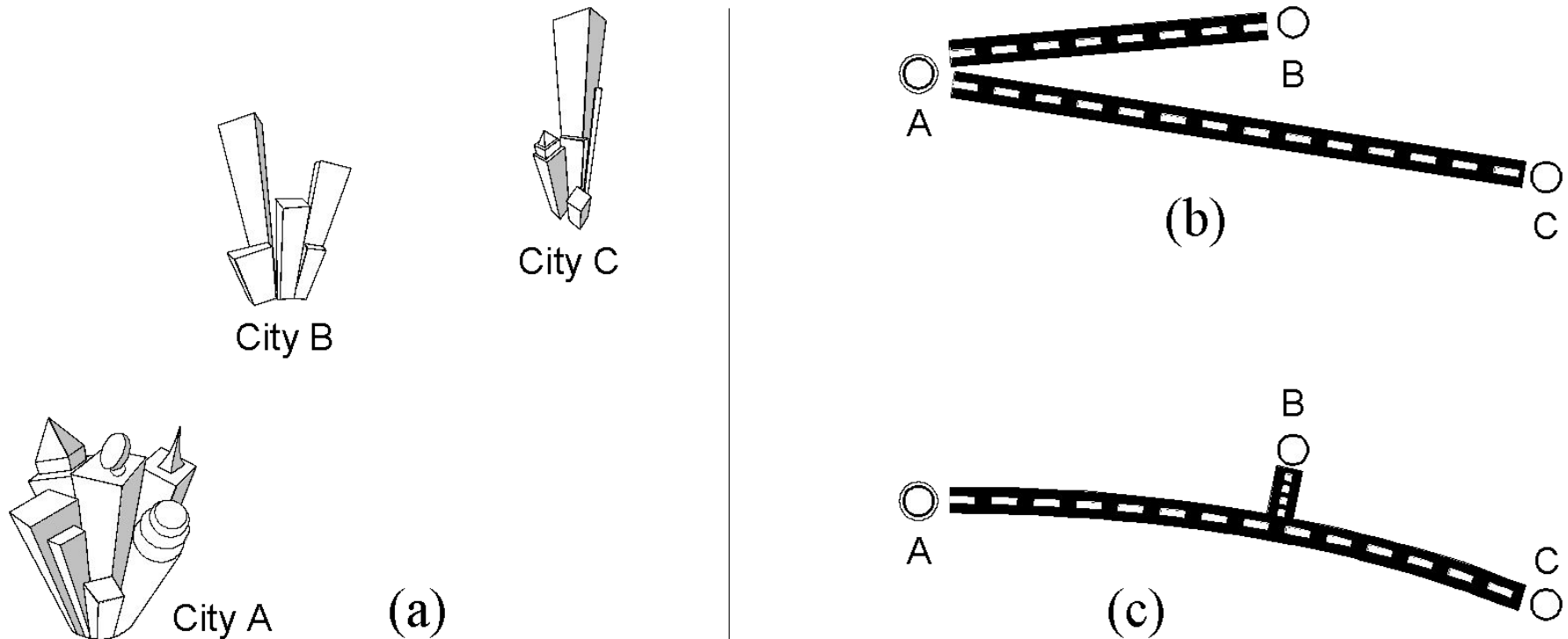


Tradeoff between Customer Flexibility and Developer Stability



- Items pulled by developers into an iteration are not subject to further customer prioritization
- Developers have a **steady goal** until the end of the current iteration
- Customer has **flexibility** to change priorities in response to changing market forces

How To Combine the Part Sizes?



Costs are not always additive

But, solution (c) is not necessarily “cheaper” than (b) ...

Requirement Specification

- Natural language Specification
- Structured Specifications
- Use Cases
- Software Requirements Document

Requirements Validation

- **Validation Checks** – checks that the requirements reflect the real needs of the system users.
- **Consistency Checks** – requirements in the document should not conflict.
- **Completeness Checks** – requirements document should include the requirements that define all the functions and constraints intended by the system user.
- **Realism Checks** – whether the requirements can be implemented within the proposed budget for the system.
- **Verifiability** – to reduce the potential for dispute between the customer and the contractor, the system requirements should always be written so that they are verifiable.

Requirements Change Management



Thank you!

